

A night photograph of a railway track. The tracks run from the foreground into the distance, with a bright light reflecting off the rails. The background shows industrial structures and a train on a side track.

# User Centered API Versioning

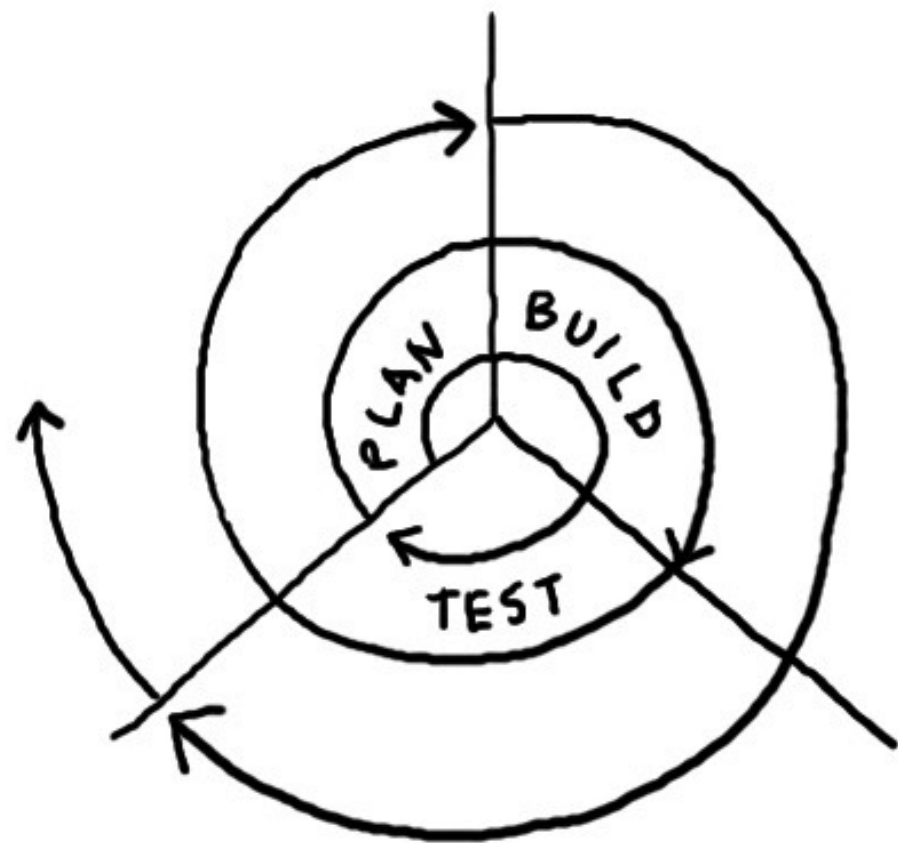
# Niall Burkley

@niallburkley





**Meltwater API**



ITERATIVE DEVELOPMENT



**An API is a contract**

# Changing our API





# Versioning

/api/**v1**/documents

/api/**v2**/documents

/api/**v3**/documents

[Developer](#)[Use cases](#)[Products](#)[Docs](#)[More](#)

Search all documentation...

# General

## Basics

## Accounts and users

## Tweets

## Direct Messages

## Media

## Trends

## Geo

## Ads

[General](#)[Analytics](#)[Audiences](#)[Campaign Management](#)[Creatives](#)[Overview](#)[Guides](#)

## Migration Guide; v0 => v1

We recommend you start moving away from analytics v0 endpoints to the newly introduced analytics v1 endpoints as soon as possible. As per our [Version 1 Announcement](#), v0 will be deprecated on *June 30, 2016*.

## Consolidation of analytics endpoints

In version 0 of the Ads API, a separate analytics endpoint existed for each entity type, from funding instruments to promoted tweets to organic tweets. With version 1 of the API, we've consolidated these into just two endpoints - one for synchronous stats queries, and another for asynchronous stats queries. These two endpoints can be used to fetch stats for all entity types, specified using the `entity` and `entity_ids` parameters. The synchronous endpoint will return smaller batches of data ideal for real-time campaign optimizations. The asynchronous endpoint is intended for larger queries of complex data, ideal for generating reporting or historical backfills.

**An alternative?**

**Stable API**

# Documentation

**Easy to Upgrade**

**Don't make me think**

**Best for us?**

**Easy to change**

**Easy to maintain**

**Incentive to upgrade**

**Support all the versions!**

A photograph showing three people on a narrow wooden bridge or walkway spanning a river. The person in the foreground is wearing a white long-sleeved shirt, a dark blue life vest with 'Polaris' written on it, and a bright yellow helmet. They are leaning forward. In the middle ground, a person in a red and white shirt, a white helmet, and a life vest is standing and holding onto a rope. In the background, a person in a black long-sleeved shirt, a blue helmet, and a life vest is sitting on a red kayak. The bridge is made of wooden planks and is surrounded by dense green foliage on the left and a turbulent river with white water rapids on the right. The text 'What!?' is overlaid in the center of the image.

What!?



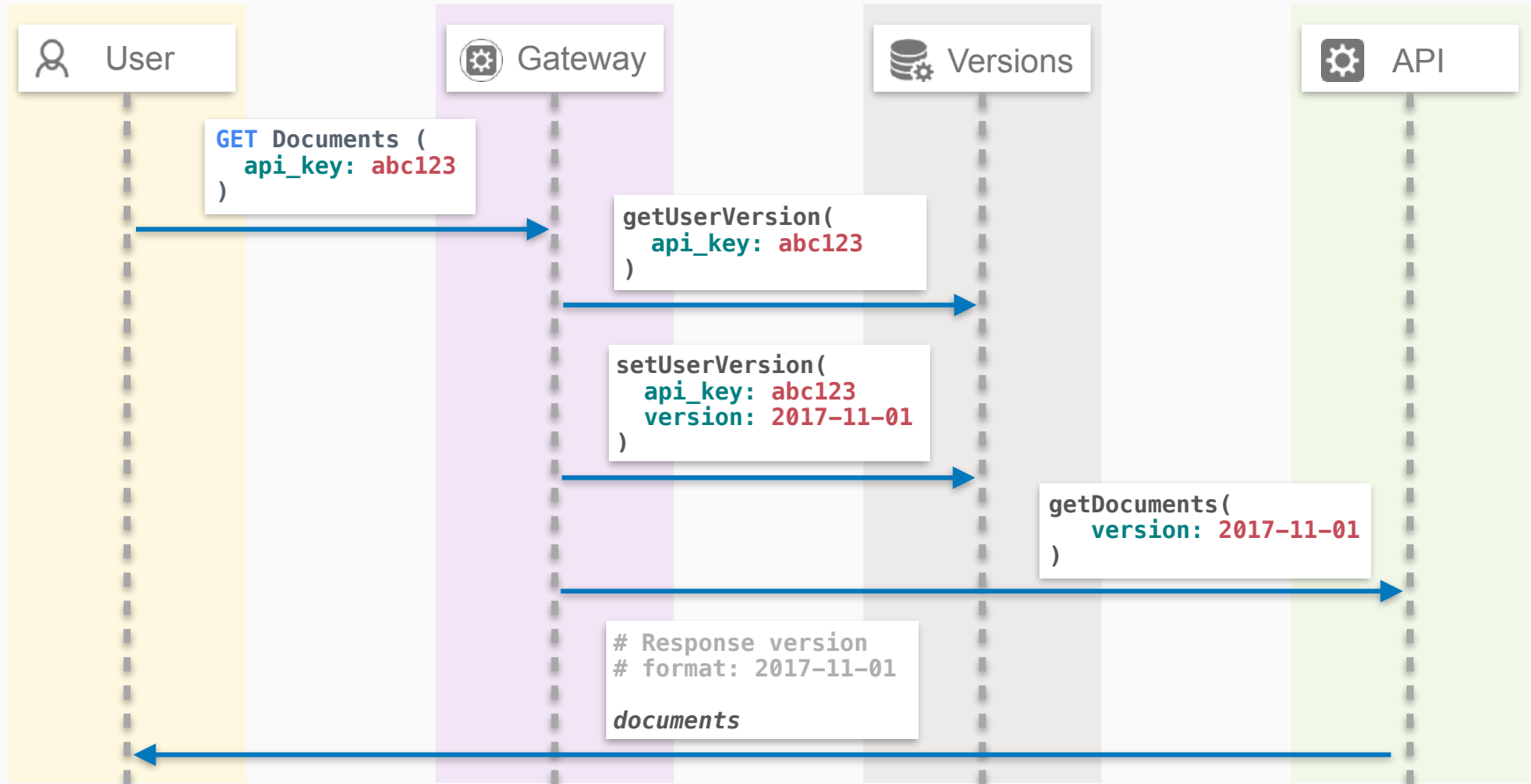
## APIs as infrastructure: future-proofing Stripe with versioning

[Brandur Leach](#) on August 15, 2017 in [Engineering](#)

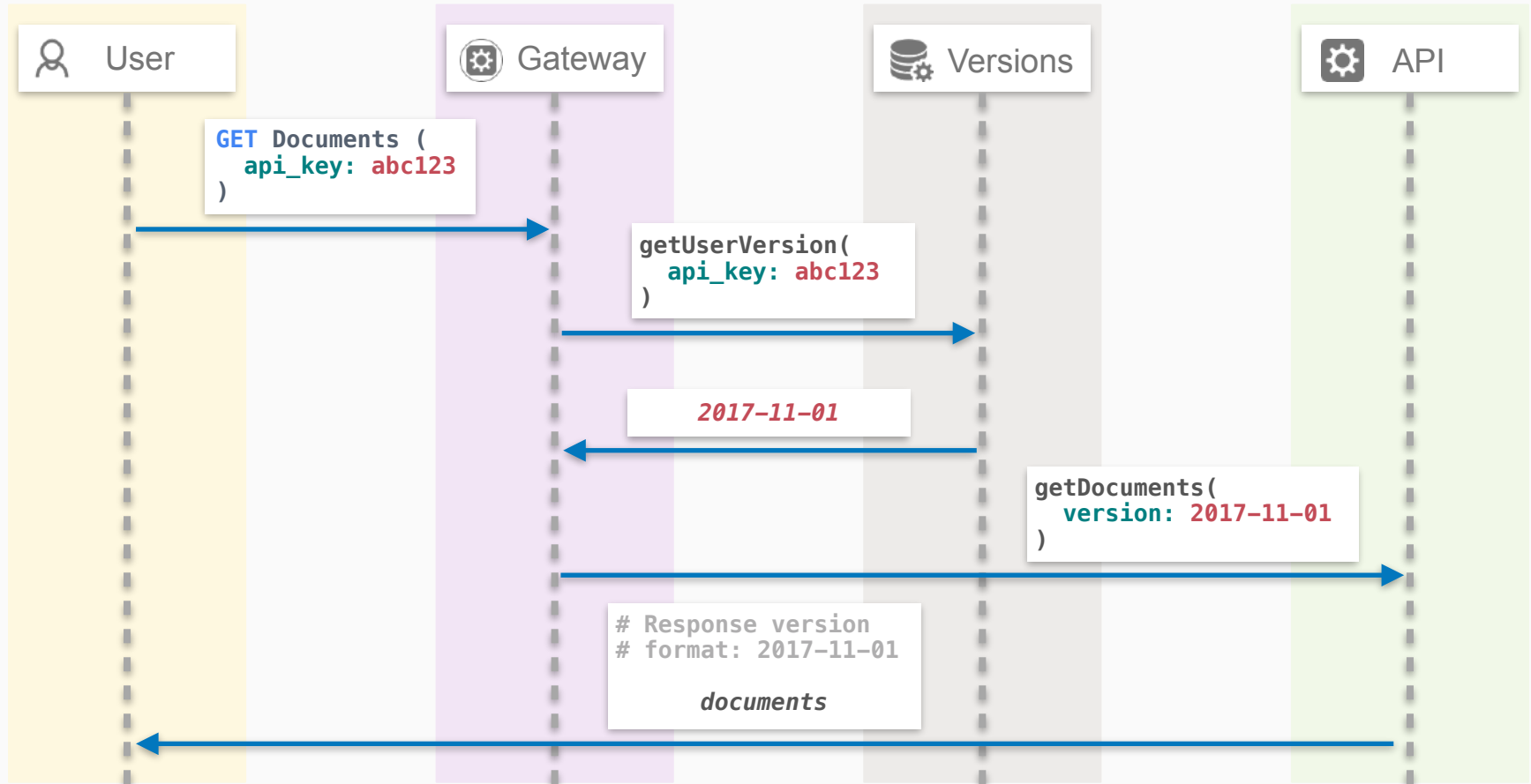
When it comes to APIs, change isn't popular. While software developers are used to iterating quickly and often, API developers lose that flexibility as soon as even one user starts consuming their interface. Many of us are familiar with how the Unix operating system evolved. In 1994, *The Unix-Haters Handbook* was published containing a long list of missives about the software—everything from overly-cryptic command names that were optimized for Teletype machines, to irreversible file deletion, to unintuitive programs with far too many options. Over twenty years later, an overwhelming majority of these complaints are still valid even across the dozens of modern derivatives. Unix had become so widely used that changing its behavior would have challenging implications. For better or worse, it established a contract with its users that defined how Unix interfaces behave.

**How does this work for the  
user?**

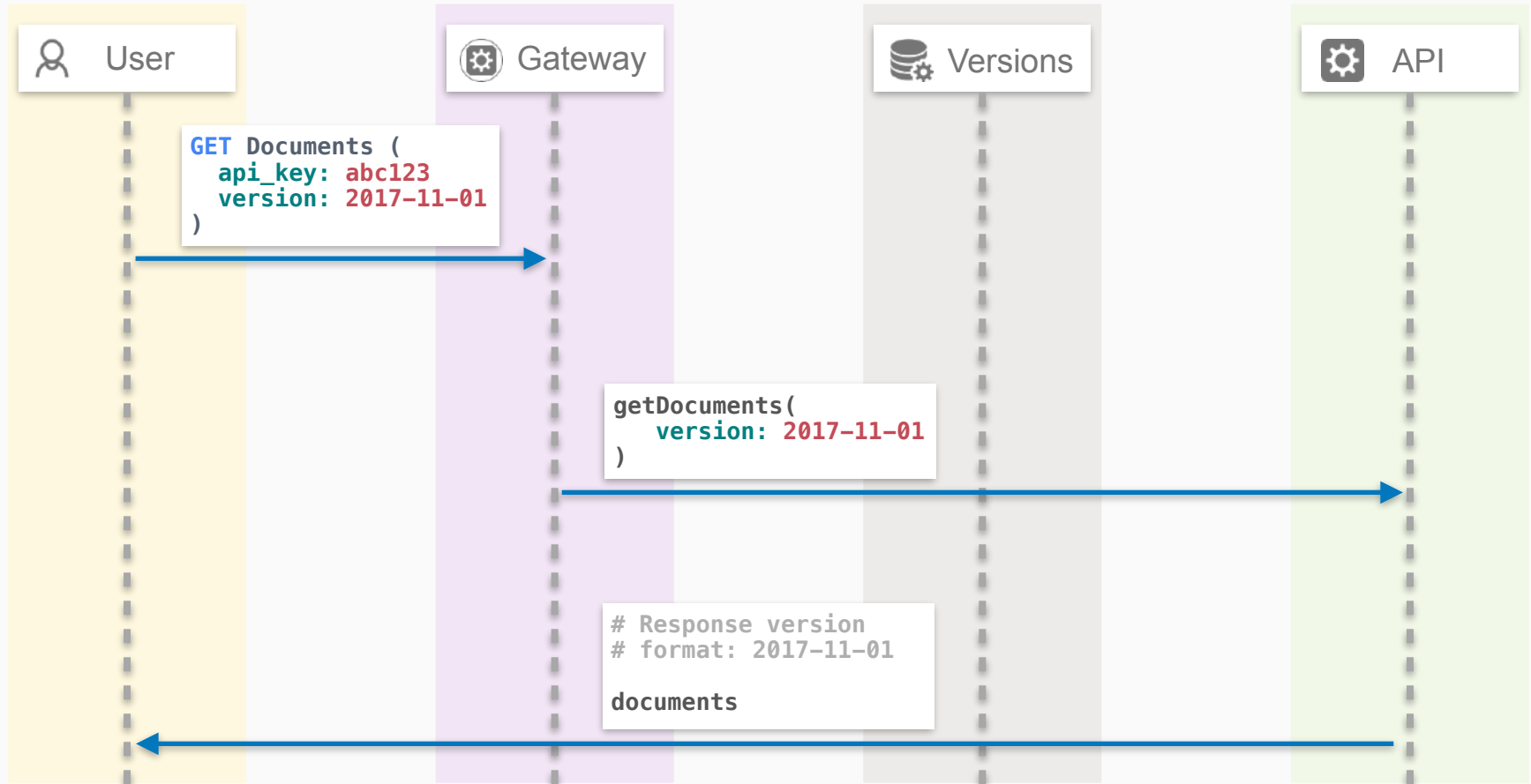
# User's first API Call



# Existing User's API Call



# Existing User's API Call - custom version

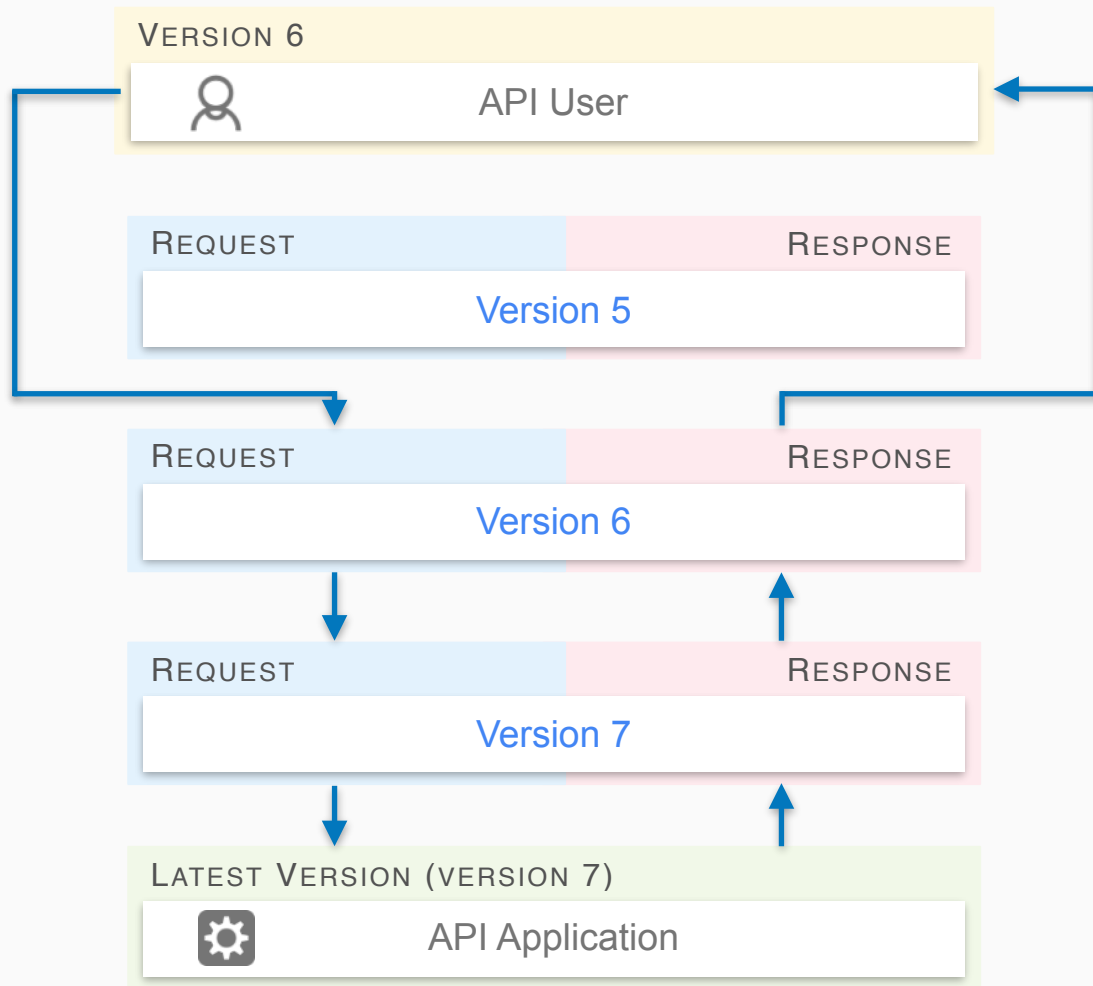


**How to build it?**

**Keep it current**

# Release rolling versions

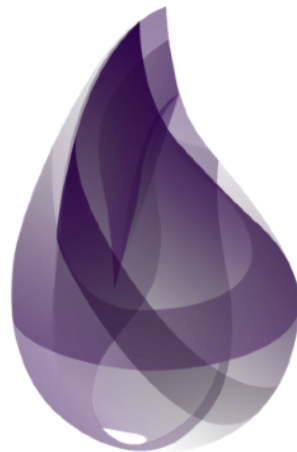
# Versions as transformations



# Implementation

$\lambda$

+



connection

|> endpoint

|> router

|> pipeline

|> controller

connection

|> endpoint

|> **plug**

|> router

|> **plug**

|> pipeline

|> controller

connection

|> endpoint

|> **authentication**

|> router

|> **apply\_version**

|> pipeline

|> controller

# Sample application

```
curl -X GET -G http://0.0.0.0:4000/api/todos --data 'size=2'
```

```
curl -X GET http://0.0.0.0:4000/api/todos --data 'size=2'
```

```
{  
  "data": [  
    {  
      "title": "Build Sample App",  
      "id": 1,  
      "description": "Put together a sample app for versioning"  
    },  
    {  
      "title": "Add documentation",  
      "id": 2,  
      "description": "Write up some documentation"  
    }  
  ]  
}
```

```
curl -X GET -G http://0.0.0.0:4000/api/todos \  
-data 'size=2'
```

```
curl -X GET -G http://0.0.0.0:4000/api/todos \  
-data 'page_size=2'
```

OLD VERSION VERSION



API User

OLD REQUEST FORMAT

```
%Plug.Conn{params: %{"size" => size}}
```

CURRENT REQUEST FORMAT

```
%Plug.Conn{params: %{"page_size" => size}}
```

LATEST VERSION



API Application

Transform  
Request



```
defmodule TodosWeb.Plugs.ModifyRequest do
  @behaviour Plug

  def init(opts), do: opts

  def call(%Plug.Conn{params: %{"size"=> size} = params} = conn, _) do
    updated_params =
      params
      |> Map.put("page_size", size)
      |> Map.delete("size")

    %{conn | params: updated_params }
  end

  def call(conn, _), do: conn
end
```

```
curl -X GET http://0.0.0.0:4000/api/todos
```

```
{
  "data": [
    {
      "title": "Build Sample App",
      "id": 1,
      "description": "Put together a sample app for versioning"
    },
    {
      "title": "Add documentation",
      "id": 2,
      "description": "Write up some documentation"
    }
  ]
}
```

...

{

{

"title": "Add documentation",

"id": 2,

"description": "Write up documentation"

}

}

...

...

{

{

"title": "Add documentation",

"id": 2,

"details": "Write up documentation"

}

}

...

LATEST VERSION



API Application



CURRENT RESPONSE FORMAT

```
{  
  "title": "Add documentation",  
  "id": 2,  
  "details": "Write up documentation"  
}
```



OLD RESPONSE FORMAT

```
{  
  "title": "Add documentation",  
  "id": 2,  
  "description": "Write up documentation"  
}
```



OLD VERSION VERSION



API User

Transform  
Response



```
defmodule TodosWeb.Plugs.TransformResponse do
  @behaviour Plug

  def init(opts), do: opts

  def call(%Plug.Conn{resp_body: body} = conn, _opts) do
    Plug.Conn.register_before_send(conn, fn conn ->
      transform_description(conn)
    end)
  end

  def call(conn, _), do: conn

  defp transform_description(%Plug.Conn{resp_body: body} = conn) do
    end
end
```

```
defmodule TodosWeb.Plugs.TransformResponse do
  @behaviour Plug

  def init(opts), do: opts

  def call(%Plug.Conn{resp_body: body} = conn, _opts) do
    Plug.Conn.register_before_send(conn, fn conn ->
      transform_description(conn)
    end)
  end
  def call(conn, _), do: conn

  defp transform_description(%Plug.Conn{resp_body: body} = conn) do
    end
  end
end
```

```
defp transform_description(%Plug.Conn{resp_body: body} = conn) do
  json_body = Poison.decode!(body)

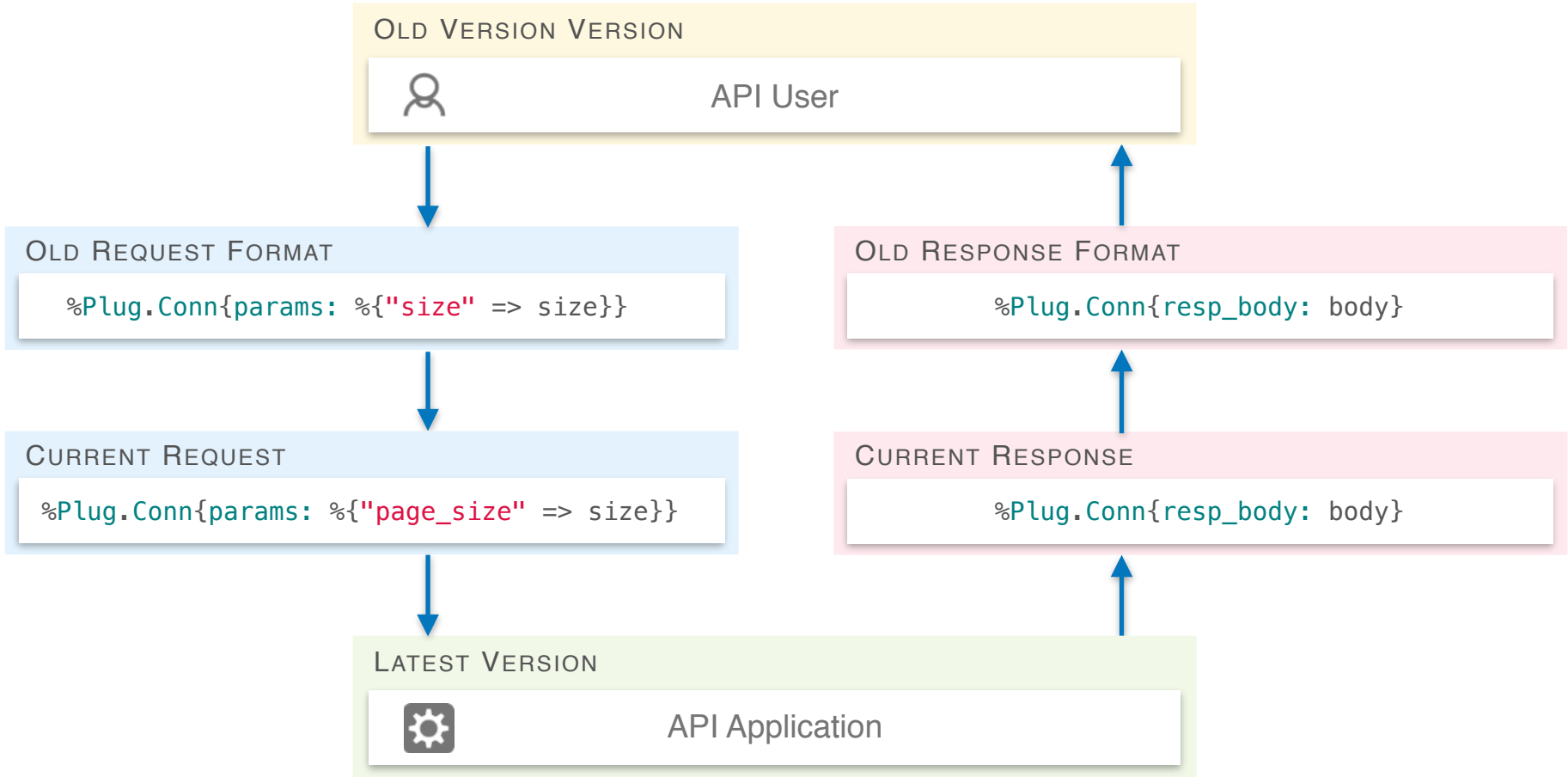
  transformed_data =
    json_body["data"]
    |> Enum.map(fn(item) ->
    |> Map.put("details", item["description"])
    |> Map.delete("description")
  end)

  %{conn | resp_body: Poison.encode!(%{json_body | "data" => transformed_data})}
end
```

Check this for a better regex for swapping out the value

```
defp transform_description(%Plug.Conn{resp_body: body}) do
  transformed_body =
    body
    |> to_string
    |> String.replace("\"description\":", "\"details\":")

  %{conn | resp_body: transformed_body }
end
```



```
defmodule TodosWeb.Change do
```

```
  @doc """
```

```
  Transforms the request on the way into the application.  
  """
```

```
  @callback transform_request(Plug.Conn.t) :: Plug.Conn.t
```

```
  @doc """
```

```
  Registers callback to transform response on the way out  
  of the application  
  """
```

```
  @callback transform_response(Plug.Conn.t) :: Plug.Conn.t
```

```
end
```

```
defmodule TodosWeb.Changes.Versions do

  ...

  @all_versions %{
    "2017-10-02" => [
      TodosWeb.Changes.RevertMultipleAuthors
    ],
    "2017-10-03" => [
      TodosWeb.Changes.RemoveDocumentLocation,
      TodosWeb.Changes.RenameSourceId
    ],
    "2017-10-04" => [
      TodosWeb.Changes.ResetSourceReachDefault
    ]
  }

  ...

end
```

```
defmodule TodosWeb.Changes.Versions do
```

```
...
```

```
def changes_for(requested_version) do
```

```
  @all_versions
```

```
  |> versions_since(requested_version)
```

```
  |> Keyword.values
```

```
  |> List.flatten
```

```
end
```

```
defp versions_since(versions, requested_version) do
```

```
  Enum.filter(versions, fn({version_date, _changes}) ->
```

```
    requested_version <= version_date
```

```
  end)
```

```
end
```

```
...
```

```
end
```

connection

|> endpoint

|> authentication

|> router

|> **apply\_version**

|> pipeline

|> controller

```
defmodule TodosWeb.Plugs.ApplyVersion do
  @behaviour Plug

  def init(opts), do: opts

  def call(conn, _) do
    # 1. get request version

    # 2. get changes for version

    # 3. apply request changes

    # 4. apply response changes
  end
end
```

```
defmodule TodosWeb.Plugs.ApplyVersion do
  @behaviour Plug

  def init(opts), do: opts

  def call(conn, _) do
    changes =
      get_req_header(conn, "x-api-version")
      |> List.first()
      |> TodosWeb.Versions.changes_for()

    # apply request changes
    Enum.reduce(changes, conn, fn change, conn ->
      change.transform_request(conn)
    end)

    # apply response changes
    Enum.reduce(changes, conn, fn change, conn ->
      Plug.Conn.register_before_send(conn, fn conn ->
        change.transform_response(conn)
      end)
    end)

  end
end
```

```
defmodule TodoAPI.Plugs.ApplyVersion do
  @behaviour Plug

  def init(opts), do: opts

  def call(conn, _) do
    changes = # should handle invalid versions
      get_req_header(conn, "x-api-version")
      |> List.first()
      |> TodoAPI.Versions.changes_for()

    # apply request changes
    Enum.reduce(changes, conn, fn change, conn ->
      change.transform_request(conn)
    end)

    # apply response changes
    Enum.reduce(changes, conn, fn change, conn ->
      Plug.Conn.register_before_send(conn, fn conn ->
        change.transform_response(conn)
      end)
    end)
  end
end
```

```
defmodule TodoAPI.Plugs.ApplyVersion do
  @behaviour Plug

  def init(opts), do: opts

  def call(conn, _) do
    changes =
      get_req_header(conn, "x-api-version")
      |> List.first()
      |> TodoAPI.Versions.changes_for()

    # apply request changes
    Enum.reduce(changes, conn, fn change, conn ->
      change.transform_request(conn)
    end)

    # apply response changes
    Enum.reduce(changes, conn, fn change, conn ->
      Plug.Conn.register_before_send(conn, fn conn ->
        change.transform_response(conn)
      end)
    end)
  end
end
```



This repository

Search

Pull requests

Issues

Marketplace

Explore



Nebo15 / multiverse

Watch

3

★ Unstar

28

🍴 Fork

4

&lt;&gt; Code

🔔 Issues 0

🔗 Pull requests 0

📊 Insights

Elixir package that allows to add compatibility layers via API gateways. <https://hex.pm/packages/multiverse>

api

plug

elixir

hex

versioning

gateways

elixir-lang

📶 51 commits

🌿 1 branch

📦 7 releases

👤 2 contributors

📄 MIT

Branch: master

New pull request

Create new file

Upload files

Find file

Clone or download



AndrewDryga Format the code with Elixir 1.6 formatter

Latest commit 83b89dc 13 days ago

📖 README.md

## Multiverse

deps



downloads 45/week

hex v1.0.0

license MIT

build passing

coverage 100%

ebert 12 Issues

This plug helps to manage multiple API versions based on request and response gateways. This is an awesome practice to hide your backward compatibility. It allows to have your code in a latest possible version, without duplicating controllers or models.

**Stable API**

**Documentation**

**Don't make me think**

**Easy to upgrade**

## What makes an API Maintainer happy?

**Easy to change**

**Keep it manageable**

**Incentive to upgrade**

- Add some performance tests
- How did we implement things?
  - Microservice architecture, internal libraries
  - Talking about experiences is better

# Why doesn't everyone do this?

# Conclusion

@niallburkley | github.com/nburkley | niallburkley.com

# Danke!



<http://underthehood.meltwater.com/>